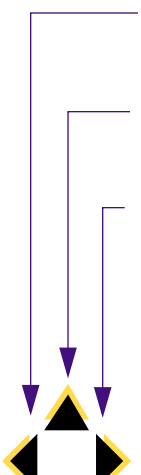
MW Debug Introduction

Welcome to CodeWarrior.

This tutorial is designed to introduce the basic features and functionality of the MW Debug program. MW Debug is a stand-alone application that enables you to debug your software and is a major part of the CodeWarrior Integrated Development Environment (IDE).

Using the Controls

Use these three controls to move through this tutorial.



Previous

Click to move to the previous page.

Return to Topics

Click to return to the main topics page.

Next

Click to move to the next page.

Adobe Acrobat

The standard Adobe Acrobat commands are also available on the View menu.

Editor Overview

Before you get started, take a moment to review the information provided in this introduction before continuing. It may answer some of the questions you have on using this tutorial. The topics include:

- Using the Controls
- Editor Topics
- Required Knowledge
- Intended Audience
- Taking the Tour
- Where Else to Go Tutorials
- Where Else to Go Documentation

Editor Topics

This tutorial has two purposes. First, it introduces the MW Debug features. Second, it guides you through common actions you need to know to effectively use the MW Debug. This tutorial includes:

- MW Debug Components
- MW Debug Actions

Required Knowledge

This tour assumes that you have a basic understanding of using your computer system. If you are just getting started with computers, refer to the user guide that came with your computer system. Once you feel comfortable with using your computer, come back and start again.

NOTE: The CodeWarrior tutorials are designed to teach you how to use CodeWarrior to successfully create, compile, and debug executable code. These tutorials will not teach you how to program a computer. If you wish to learn programming, visit your local bookstore for beginning programming books, see your local community college for classes on programming, or refer to the 3rd-party documentation that may be provided on your CodeWarrior CD.

Intended Audience

The Debugger (Basic) tutorial is intended for all first-time code warriors. If you have not already taken the Guided Tour tutorial, we recommend that you take a few minutes to go through that tutorial before continuing here. The Guided Tour will give you an overview of the entire CodeWarrior IDE.

Beginning Programmers

If you are just beginning your career as a programmer then you are in for a treat. The CodeWarrior environment provides all the tools you need for any programming task, from simple applications to advanced professional-quality programs. Go through this tour, then follow the recommendations at "Where Else to Go Tutorials" to continue your explorations of CodeWarrior.

more…

Intended Audience (continued)

Experienced Programmers

After taking the Guided Tour tutorial, experienced programmers new to CodeWarrior should refer to this and the other tutorials for in-depth knowledge about the various components of the CodeWarrior environment. Many CodeWarrior Targeting manusl include basic tutorials that demonstrate how to use the tools to program for a particular platform.

Taking the Tour

The CodeWarrior tutorials are designed to help you become familiar with the CodeWarrior IDE. This tutorial consists of a series of stages. Each stage, in turn, is a series of slides, like this one. Each slide describes a feature or a task you can perform. You will gain the most benefit from these tutorials if you follow along with the CodeWarrior IDE while reading and performing the actions described in the tutorial.

To Use a Tutorial:

1. In Acrobat Reader 3.0, read the text on a slide.
2. If actions are described (example: Choose Open from the File menu), then switch to the CodeWarrior IDE.
3. In the CodeWarrior IDE, perform the action.
4. Switch back to Adobe Acrobat Reader, read the next action.

NOTE: Both Adobe Acrobat Reader and the CodeWarrior IDE must be running.

Host Conventions

Hosts: While the CodeWarrior IDE runs on several different host platforms, the functionality on these host platforms is not always identical. There are occasions when a feature is implemented for one or more hosts and not for others. This tutorial only demonstrates features found on all supported hosts.

Pictures: The IDE's visual interface on these hosts is nearly identical in all significant respects. When discussing a particular interface element such as a dialog box or window, the screenshot may come from any host. If you are using CodeWarrior on a different host, you should have no difficulty understanding the picture.

Keyboard: The default keyboard shortcuts for CodeWarrior on these platforms are very similar. However, keyboards and shortcuts do vary across host platforms. For example, a typical keyboard for a Windows machine has an Alt key, but a typical keyboard for a Mac OS computer calls the same key the Option key

more…

Host Conventions (continued)

To handle these kinds of situations, CodeWarrior documentation identifies and uses the following paired terms in the text:

- `Enter/Return` - the “carriage return” or “end of line” key. This is not the numeric keypad Enter key, although in almost all cases that works the same.
- `Backspace/Delete` - the key that (in text editing) causes the character before the insertion point to be erased (not to be confused with `Delete/Del`, the “forward delete” key)
- `Ctrl/Command` - in most cases, CodeWarrior maps the Windows Ctrl key the same way as the Mac OS Command key.
- `Alt/Option` - in most cases, CodeWarrior maps the Windows Alt key the same way as the Mac OS Option key.

For example, you may encounter instructions such as “Press `Ctrl/Command+S` to save a file,” or “Alt/Option click the Function pop-up to see the functions in alphabetical order." Use the appropriate key as it is labeled on your keyboard.

Where Else to Go Tutorials

Once you have finished this tutorial, be sure to explore the other CodeWarrior tutorials, and the documentation that accompanies this product. For an overview of the IDE, see:

- Guided Tour–overview of the CodeWarrior IDE
- Project Manager (Basic)–introduces the Project Manager
- Editor (Basic)–introduces the CodeWarrior Editor
- MW Debugger (Basic)–introduces debugging features
- Find & Replace–introduces Find and Replace features

Where Else to Go Documentation

To learn more about programming for a particular target operating system or processor, see the corresponding CodeWarrior Targeting manual. The CodeWarrior Debugger features are also documented in:

- IDE User Guide–documents IDE components, including Find and Replace.
- Debugger User Guide–documents the Metrowerks standalone debugger.
- Targeting manual for your programming target. Examples include Targeting Windows, Targeting Rhapsody, and Targeting Palm OS.

Getting Started

Phew! Now that all of the preliminaries are out of the way, its time to explore MW Debug.

MW Debug Components

The following stages identify some of the windows you will use during debugging sessions with MW Debug. These windows include:

- STAGE 1: Touring the Program Window
- STAGE 2: Touring the Browser Window
- STAGE 3: Touring the Breakpoint Window
- STAGE 4: Touring the Watchpoint Window
- STAGE 5: Touring the Variables Window
- STAGE 6: Touring the Preferences Window

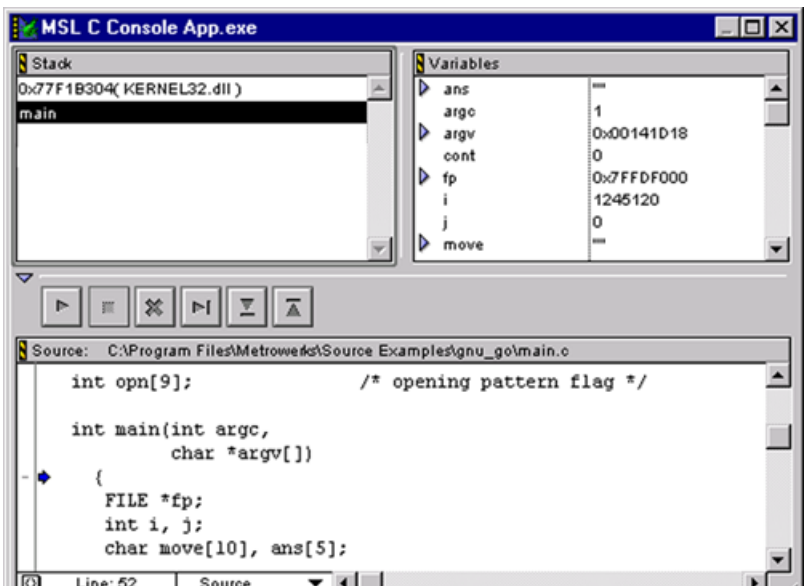
Touring the Program Window

The Program window displays debugging information about the currently executing code and the source code file associated with it. The Program window has several components associated with it, including:

- Stack Crawl Pane
- Variables Pane
- Source Pane
- Debugger Toolbar
- Function Pop-up
- Source Pop-up

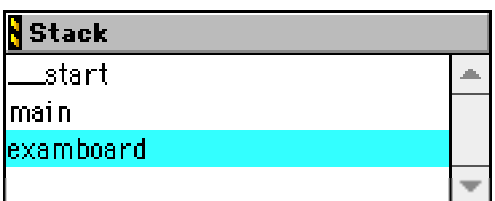
Touring the Program Window (continued)

Here's what the Program window looks like:



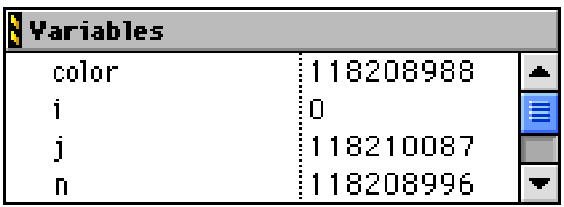
Stack Crawl Pane

The Stack Crawl pane shows the current subroutine calling chain. Each subroutine is added below the routine that called it, providing a visual map of the program's execution. In addition, the selected function's source codee appears ion the Source pane.



Variables Pane

The Variables pane displays the currently executing routine's local variables as well as any referenced global variables.



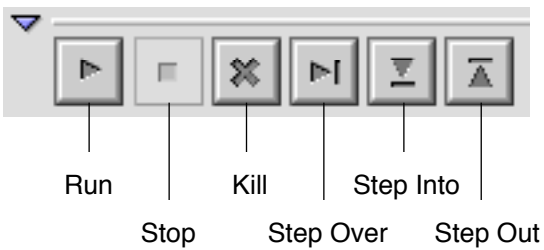
Source Pane

The Source pane displays the contents of the source code file as well as the Current-Statement Arrow, which indicates the currently executing statement in the program.

Source : Europa :Working Docs :Tutorials :...rce Examples :gnu_go :examboard.c
void examboard(int color) /* BLACK or WHITE */
/* examine pieces */
{
int i, j, n;

Debugger Toolbar

The Toolbar gives access to the MW Debug execution commands found in the Control menu.



Debugger Toolbar (continued)

The buttons on the Toolbar include:

Run–restarts execution of a project.

Stop–temporarily halts execution of a project.

Kill–kills execution of a project.

Step Over–executes a single statement, including function calls when encountered.

Step Into–executes a single statement, stepping into routine calls when encountered.

Step Out–executes statements until the end of the routine.

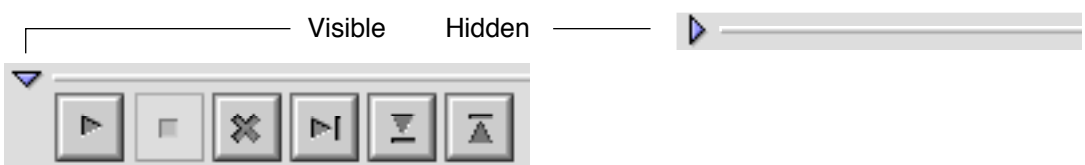
Debugger Toolbar (continued)

In this step you hide and show the Toolbar in the Program window.

To Hide or Show the Toolbar:

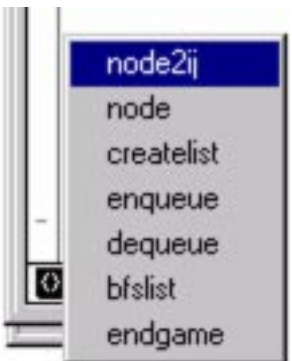
- Click the Disclosure Triangle.

  The Toolbar is hidden. Click again to show it.



Function Pop-up

The Function pop-up displays a list of all routine names in the currently executing source file. Choose a routine displays it in the Source pane.



Source Pop-up

Use the Source pop-up to specify how the Source pane displays the contents of the source code file.

The choices include:

Source–displays source code in the Source pane.

Assembly–displays assembly code in the Source pane.

Mixed–displays mixed source and assembly code in the Source pane.

Summary

This completes the Program Window stage of this tour.

In this stage you learned how to:

- Identify the Stack Crawl, Variables, and Source pane
- Locate the Toolbar buttons
- Locate and use the Function and Source pop-ups

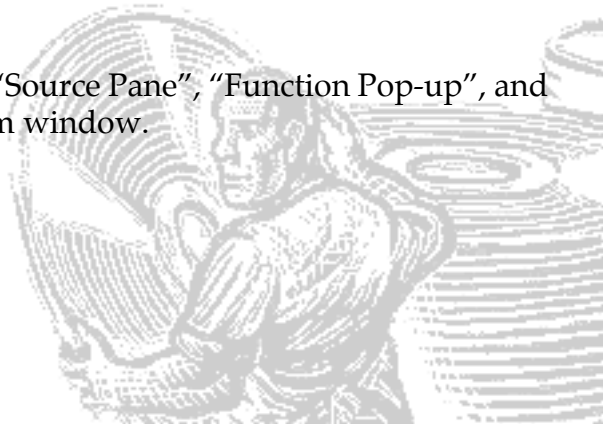
As you can see, there is a lot of information provided in the Program window. But there is another window that is equally important, the Browser window. We'll explore the Browser window in the next stage of this tour.

Touring the Browser Window

The Browser window enables you to view any file contained in the project, and to view and edit the program's global variables. It has several components associated with it, including:

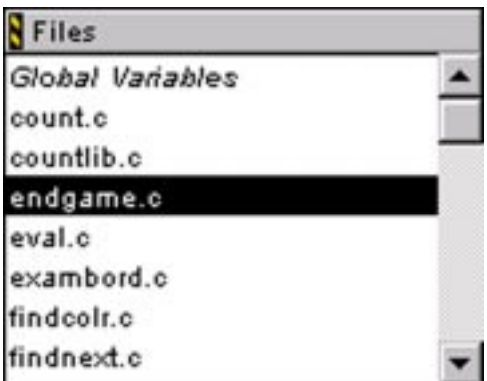
- Files Pane
- Function Pane
- Globals Pane

The Browser window also has the same "Source Pane", "Function Pop-up", and "Source Pop-up" you saw in the Program window.



Files Pane

The Files pane displays a list of all files contained in the project. The code for the selected file appears in the Source pane.



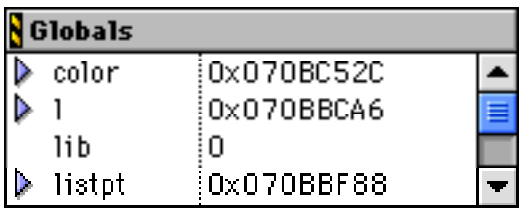
Function Pane

The Function pane displays a list of all the functions in the currently executing file. The code for the selected function appears in the Source pane.

See “Browser Window Navigation” for information on using the Function pane.

Globals Pane

The Globals pane displays any referenced global variables.



See “Changing Program Variables” for information on using the Globals pane.

Summary

This completes the Browser Window stage of this tour.

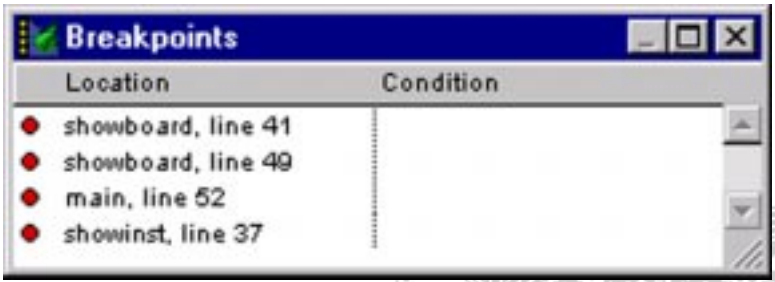
In this stage you learned how to:

- Identify the Files, Functions, Globals panes

In addition to the Program and Browser windows, there a several ancillary windows that are used by MW Debug to display specific information. Let's continue our tour ot MW Debug by looking at the Breakpoint window.

Touring the Breakpoint Window

The Breakpoint window lists all breakpoints in your project, by source file and line number.



Opening and Closing the Breakpoint Window

In this step you open and close the Breakpoint window.

To Open the Breakpoint Window:

- Choose **Show Breakpoints** from the **Window** menu.

  The Breakpoint window appears.

To Close the Breakpoint Window:

- Choose **Hide Breakpoints** from the **Window** menu, **OR**
  Click the window's close box, **OR**
  Type the keyboard shortcut Ctrl/Command+W.

  The Breakpoint window closes.

Summary

This completes the Breakpoint Window stage of this tour.

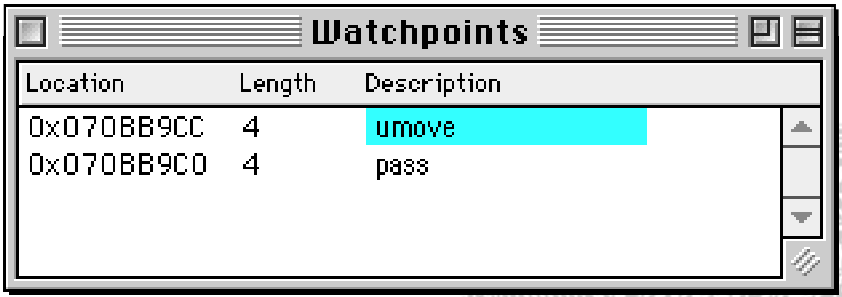
In this stage you learned how to:

- Open and close the Breakpoint window

Let's continue our tour of MW Debug with the Watchpoint window.

Touring the Watchpoint Window

The Watchpoint window displays a list of currently set watchpoints. A watchpoint is a location or region of memory that the debugger watches, alerting you when any change is made in the area.



Opening and Closing the Watchpoint Window

In this step you open and close the Watchpoint window.

To Open the Watchpoint Window:

- Choose **Show Watchpoints** from the **Window** menu.

  The Watchpoint window appears. If any watchpoints are already set, they appear within the window.

To Close the Watchpoint Window:

- Choose **Hide Watchpoints** from the **Window** menu, **OR**
  Click the window's close box, **OR**
  Type the keyboard shortcut Ctrl/Command+W.

  The Watchpoint window closes.

Summary

This completes the Watchpoint window stage of this tour.

In this stage you learned how to:

- Open and close the Watchpoint window

Using the Watchpoint window to monitor a host of different memory locations and alert you when something changes can be a real boon when debugging software. Let's continue our tour of MW Debug windows with the Variables window.

Touring the Variables Window

The Variables window displays a single variable which can be edited. In this step you open and close a Variables window.

- To Open a Variable Window
- To Close a Variable Window

To Open a Variable Window

To view any program variable in its own window, follow these steps:

To Open a Variables Window:

- Double-click a variable, **OR**
  Select a variable, then choose **Open Variable Window** from the **Data** menu.

  Try choosing the "umove" variable in the Program window's Variables pane. A separate Variables window appears with the variable umove's information.

To Close a Variable Window

To close any variable window, follow these steps:

To Close a Variables Window:

- Click the variable window's close box, **OR**
  Click the window's close box, **OR**
  Type the keyboard shortcut Ctrl/Command+W.

  The Variable window closes.

Summary

This completes the Variables Window stage of this tour.

In this stage you learned how to:

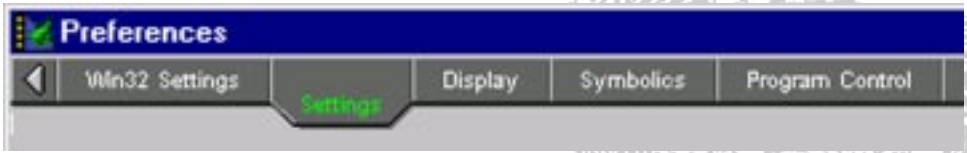
- Open and close variable windows

There's one more basic window in MW Debug that you need to be familiar with, and that is the Preferences window. In the next stage you tour the MW Debug Preferences window.

Touring the Preferences Window

The Preferences window enables you to set various options that control aspects of the Debugger's behavior. In this stage you:

- Preference Window Controls
- Opening and Closing the Preferences Window
- Modifying a Preference Setting



Preference Window Controls

The following view tabs appear in the Preferences window. You click a view tab to display its option settings.

Win32 Settings–option to specify default source code editor.

Settings–options related to the data saved between debugging sessions.

Display–options related to how tasks and variables are displayed.

Symbolics–options related to the opening, closing, and management of symbolic files.

Program Control–options related to launching, terminating, and control of the program being debugged.

Java Settings–options specific to debugging Java programs and applets.

MetroNub–options for using the MetroNub extension.

Runtime Settings–options specific to the configuration of the Windows environment.

Preference Window Controls (continued)

In addition, the Preference window has two controls that appear with every view. They are:

OK–saves changes and closes the Preferences window.

Cancel–discards all changes and closes the Preferences window.

Opening and Closing the Preferences Window

To open and close the Preferences window:

To Open the Preferences Window:

- Choose **Preferences** from the **Edit** menu.

  The Preferences window appears.

To Close the Preferences Window:

- Click **OK**.

  Any modified option settings are saved and the Preferences window closed.

  or:

- Click **Cancel**.

  The Preferences window closes without saving any changes.

Modifying a Preference Setting

You modify a debugger option to change the manner in which MW Debug behaves. In this step you change a debugger setting. See the Debugger User Guide for information on a particular option.

To Modify a Preference Setting:

1. Open the Preferences window.
2. Click the view tab containing the option.

   Click the "Settings" view tab.
3. Modify the option.

   Click "Save Expressions" to disable this feature.
4. Click **OK**.

   Your changes are saved and the Preference window closes. Click Cancel if you change your mind and don't want to save your modifications.

Summary

This completes the Preferences Window stage of this tour.

In this stage you learned how to:

- Open and close the Preference window
- Modify a preference setting

This also concludes your introduction to the basic MW Debug windows. Now that you understand what a majority of these window do, its time to actually begin using them.

MW Debug Actions

In the following sections you begin to use MW Debug for simple debugging tasks. The tasks introduced in this section include:

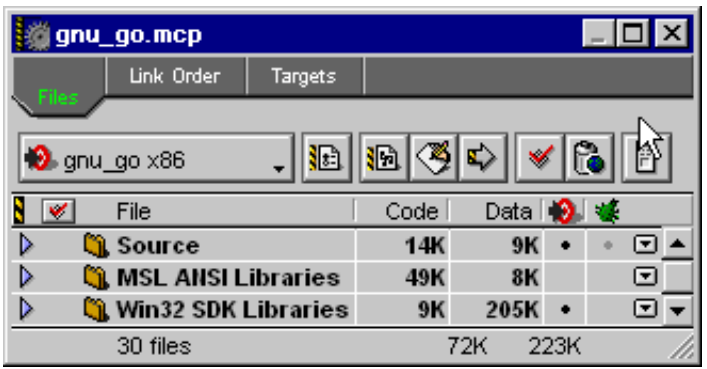
- STAGE 7: Enabling Debugging
- STAGE 8: Launching MW Debug
- STAGE 9: Operating MW Debug
- STAGE 10: Navigating Your Source Code
- STAGE 11: Using Breakpoints for Debugging
- STAGE 12: Using Watchpoints for Debugging
- STAGE 13: Using Variables for Debugging

Getting Ready

You will use the `gnu_go.mcp` project file throughout this tutorial. If you don't already have it open, please open the project now.

The `gnu_go.mcp` project file is located in the `Source Examples` folder inside the `Core Tutorials` folder.

If you don't know how to open a project file, see the Guided Tour tutorial.



Enabling Debugging

In this step you enable debugging within the CodeWarrior IDE.

The topics include:

- Enabling Debugging
- Debugging All Files
- Controlling Symbolics Per File

Enabling Debugging

When you enable debugging within the CodeWarrior IDE, the compiler and linker add symbolic information to your code. Symbolics allow the debugger to keep track of theat the source code corresponds to the object code running in the debugger. This in turn, enables you to control your software in MW Debug.

To Enable the Debugger:

- Choose **Enable Debugger** from the **Project** menu.

  The options in the target's linker panel are set for debugging. All files in the target are marked with a bullet character in the Debug column of the Project window. This means that symbolics will be created for every file.

NOTE: You might see an alert dialog asking you to "Enable debugging for target." Click Yes to automatically set all the debugging options for your project.

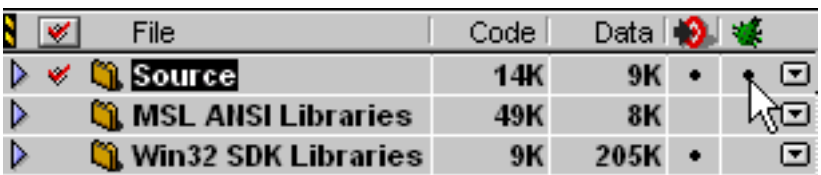
Debugging All Files

Once the Debugger is enabled, you have the option of debugging all the files in a project, or just a select few. In this step you setup the `gnu_go.mcp` project to debug all files.

To Debug All Files:

- Click in the **Debug** column next to the Group name.

  Try clicking the "`Sources`" group in the Project window. Bullet characters appear in the Debug column next to each file in the `Sources` group.



Controlling Symbolics Per File

Once debugging is enabled, you have the option of generating symbolics for all the files in a project, or just a select few. You must have symbolics on for a file if you intend to debug code in that file. In this step you use the `gnu_go.mcp` project to learn how to turn symbolics on or off for a file or group of files.

To Control Symbolics Per Group or File:

- Click in the **Debug** column next to the file or group name.

  Click the "`Sources`" group in the Project window. A bullet already presetn disappears for the group, and for any file in the group. Click again to turn symbolics back on. Feel free to experiment with individual files. However, before you leave this step, make sure symbolics are on for all the source files.

NOTE: A gray bullet character in the Debug column of a group name indicates that only some of the files in that group have been setup for debugging.

Summary

This completes the Enabling Debugging stage of this tour.

In this stage you learned how to:

- Enable debugging in CodeWarrior
- Set all files for debugging
- Enable and disable symbolics for groups or files

With the Debugger enabled in CodeWarrior, it's time to move on to a real debugging session. We start by launching MW Debug.

Launching MW Debug

In this step you launch MWDebug from the CodeWarrior IDE.

The topics covered include:

- Launching from CodeWarrior
- Launching MW Debug on Windows
- Launching MW Debug on Mac OS

Launching from CodeWarrior

Once debugging has been enabled within CodeWarrior, you can debug your project. In this step you launch MWDebug from CodeWarrior.

From CodeWarrior

- Choose **Debug** from the **Project** menu, **OR**
  Type the keyboard shortcut F5/Command+R.

  MW Debug launches and the Program and Browser windows appear.

NOTE: The Debug command is only available for debugging executable code, like applications.

Launching MW Debug on Windows

MW Debug is a stand-alone application that can be operated independent of the CodeWarrior IDE. You then open a symbolics file (.exe or `.SYM`) to debug the project associated with it. In this step you launch MWDebug and open a symbolics file.

To Launch MWDebug on Windows:

1. Double-click the MW Debug.exe application icon, **OR**
   Click Start, then locate and click CodeWarrior Debugger.

   The MW Debug application is launched and an open file dialog box appears.
2. Locate the program's `.exe` file to open.

   Open the "`gnu_go_x86.exe`." file.
3. Click **Open**.

   The Program and Browser windows appear.

Launching MW Debug on Mac OS

To Launch MWDebug on the Mac OS:

1. Double-click the MW Debug.exe application icon, **OR**
   Select MW Debug, then choose **Open** from the Finder's **File** menu.

   The MW Debug application is launched and an open file dialog box appears.
2. Locate the `.SYM` file to open.

   Open the "`gnu_go_68k.SYM`." file.
3. Click **Open**.

   The Program and Browser windows appear.

Summary

This completes the Launching MW Debug stage of this tour.

In this stage you learned how to:

- On Windows, to launch MW Debug and open the program's `.exe` file
- On the Mac OS, to launch MW Debug and open a symbolics file

With MW Debug launched and a project's symbolics file open, you are now ready to "walk" your project's source code. You explore that next in Operating the Debugger.

Operating MW Debug

With a project `SYM` file loaded its now possible to debug your program's symbolic file. You control the program's execution in MW Debug using commands present on the Toolbar and available on the Control menu.

The topics covered in this stage include:

- Stepping Over a Statement
- Stepping Into a Routine
- Stepping Out of a Routine
- Running Code
- Stopping Code
- Killing Code

Stepping Over a Statement

Step Over executes a single statement and advances to the next statement in the code. If the statement includes a routine call, the routine is executed and control returns to the next statement after the routine call.

To Step Over a Statement:

- Click the **Step Over** icon button, **OR**
  Choose **Step Over** from the **Control** menu, **OR**
  Type the keyboard shortcut Ctrl/Command+S.

  The current statement is executed and control advances to the next statement as indicated by the current-statement arrow. Repeat the Step Over command until you reach the `showinst()` statement in `main.c`.

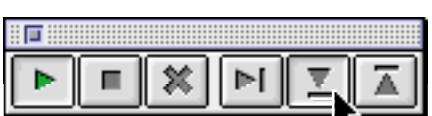
Stepping Into a Routine

You **Step Into** a routine to follow program execution into the named routine instead of executing it completely and stepping to the next statement. If the statement is not a routine, execution proceeds exactly as the **Step Over** command.

To Step Into a Routine:

- Click the **Step Into** icon button, **OR**
  Choose **Step Into** from the **Control** menu, **OR**
  Type the keyboard shortcut Ctrl/Command+T.

  Do this now to step into the `showinst()` function. Note that the Source pane now displays the `showinst()` routine.



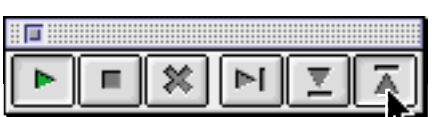
Stepping Out of a Routine

You **Step Out** of a routine to execute the statements in the routine until it returns to its caller, that is, one level up the routine call chain.

To Step Out of a Routine:

- Click the **Step Out** icon button, **OR**
  Choose **Step Out** from the **Control** menu, **OR**
  Type the keyboard shortcut Ctrl/Command+U.

  All the statements remaining in the routine are executed, control returns to the caller, then execution stops. Do this now to step out of the `showinst()` routine.



Running Code

You run a program to start or resume execution of a stopped program. In this step you Run the `gnu_go_x86.mcp/gnu_go_68k.mcp` program.

To Run a Project:

- Click the **Run** icon button, **OR**
  Choose **Run** from the **Control** menu, **OR**
  Type the keyboard shortcut Ctrl/Command+R.

  The program begins executing until a **Stop** or **Kill** command is issued, or a breakpoint is encountered in the source code. Do this now. The program begins executing.

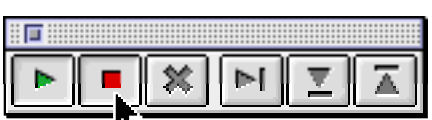
Stopping Code

You stop a program to temporarily halt program execution and allow time to examine local and global variables, registers, and other attributes of the running program. In this step you Stop the `gnu_go_x86.mcp/gnu_go_68k.mcp` program.

To Stop a Project:

- Click the **Stop** icon button, **OR**
  Choose **Stop** from the **Control** menu, **OR**
  Type the keyboard shortcut Ctrl/Command+. (period).

  The program halts executing until a **Run** command restarts the program from the current position. Do this now. The program stops executing.



Killing Code

You kill a program to terminate program execution. In this step you kill the `gnu_go_x86.mcp/gnu_go_68k.mcp` program.

To Kill a Project:

- Click the **Kill** icon button, **OR**
  Choose **Kill** from the **Control** menu, **OR**
  Type the keyboard shortcut Ctrl/Command+K.

  The program is killed. A **Run** command restarts program execution from the beginning. Do this now. The program terminates. Use Run to restart the program.

Summary

This completes the Operating MW Debug stage of this tour.

In this stage you learned how to:

- Run, stop, and kill projects
- Step over, into, and out of routines

With basic control of your project in hand, the next step is to navigate through the various files that make up your project. We explore that in the next stage.

Navigating Your Source Code

This stage introduces several methods of navigating your source code while debugging. These navigational methods include:

- Linear Navigation
- Call Chain Navigation
- Browser Window Navigation
- Find Navigation

Linear Navigation

You use the **Step Over**, **Step In**, and **Step Out** commands to "walk" your code in a linear fashion. This is helpful for observing a program as each line of code is executed.

See "Stepping Over a Statement", "Stepping Into a Routine", and "Stepping Out of a Routine" for more information.

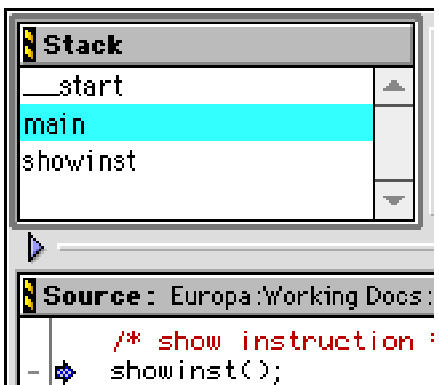
Call Chain Navigation

You use the Stack Crawl pane in the Program window to "walk" the routine call chain used to reach the currently executing routine. Each routine is added to the bottom of the list when called, with the currently executing routine at the very bottom.

To Select a Routine:

- Select a routine name, **OR**
  Use the Up and Down arrows.

  Select "`showinst`." The routine appears in the source code pane of the window.



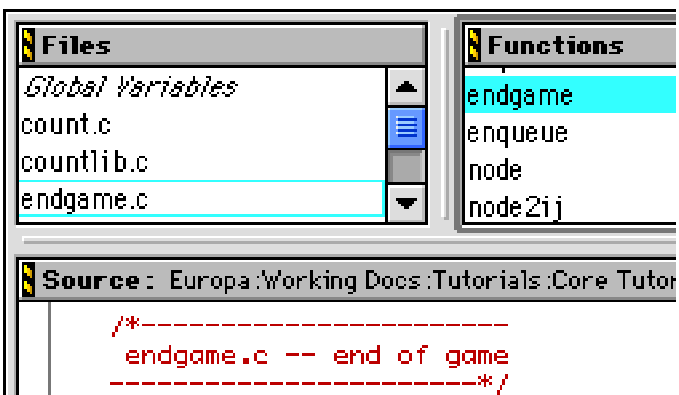
Browser Window Navigation

You use the Browser window to jump to any location in the program. Select a file in the Files pane, and a routine name in the Functions pane, or use the Functions pop-up to locate any routine in the program.

To Browse by File:

- Select a file name in the Files pane.

  Select "`endgame.c`." The selected file appears in the window's source code pane.



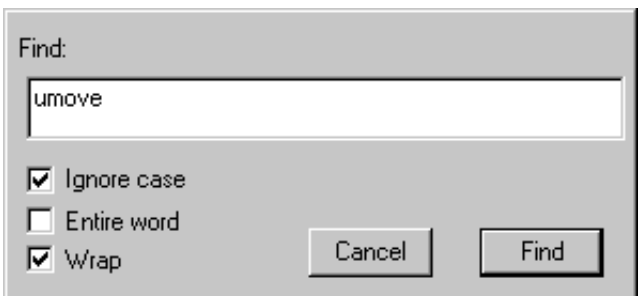
Find Navigation

You use the Find command to search for text in the source pane of the Program or Browser windows.

To Use Find:

1. Choose **Find** from the **Edit** menu, **OR**
   Type the keyboard shortcut Ctrl/Command+F.

   The Find dialog box appears.



Find Navigation (continued)

2. Enter the search text.

   Enter "`umove`" in the Find field.

3. Set the search options.

   Enable the **Ignore Case**, **Entire Word**, or **Wrap** checkbox options to set the search parameters.

4. Click **Find**.

   Find searches the current file for matches to the search text. Use Ctrl/Command+G to repeat the search.

Summary

This completes the Navigating stage of this tour.

In this stage you learned how to:

- Navigate your code linearly, by Call Chain, Browser, and using Find

Navigating enables you to effectively move about your source code, quickly finding routines via a variety of methods. In the next stage you explore break-points.

Using Breakpoints for Debugging

This stage introduces the use of breakpoints. Breakpoints allow you to stop project execution at a precise line or under a specific condition, enabling you to examine the state of the project variables, expressions, and structures.

The topics on breakpoints include:

- Setting a Breakpoint
- Viewing a Breakpoint
- Setting a Temporary Breakpoint
- Setting a Conditional Breakpoint
- To Disable a Breakpoint
- Saving Breakpoints
- Clearing a Breakpoint

Setting a Breakpoint

You set a breakpoint to halt program execution at a specific line in your program. In this step you set a breakpoint in the `showboard()` routine in the `showbord.c` file.

To Set a Breakpoint:

1. Identify where to set the breakpoint.

   Locate line #41 (an open brace "`{`") in the file `showbord.c`.

2. Click the breakpoint dash in the column next to the line.

   A red breakpoint circle appears indicating the presence of a breakpoint. Repeat with Line #49 (`ii = 19 - i;`) in the same file.

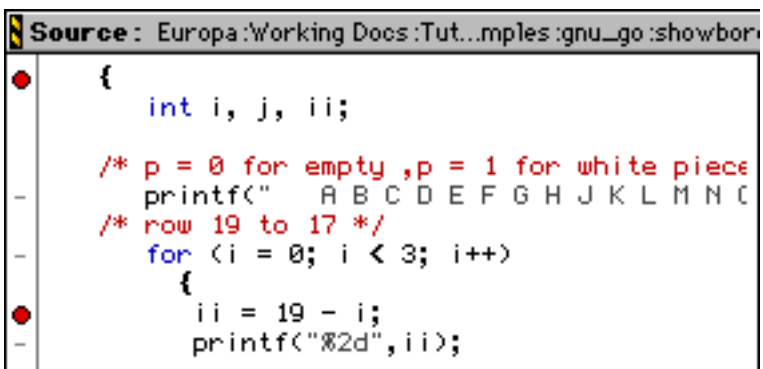
3. Choose **Run** from the **Control** menu, **OR**
   Click the **Run** button in the Toolbar, **OR**
   Type the keyboard shortcut Ctrl/Command+R.

   The program runs until the first breakpoint is encountered.

Setting a Breakpoint (continued)

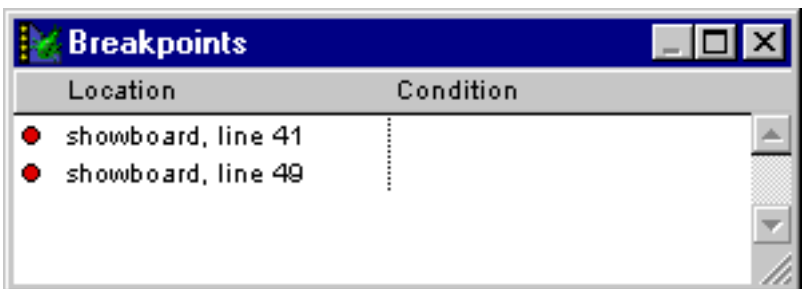
4. Run the code again.

   Click the Run button in the Toolbar. The program runs until the second breakpoint is encountered.


Source: Europa:Working Docs:Tut...mples:gnu_go:showbor
{
int i, j, ii;
/* p = 0 for empty ,p = 1 for white piece
printf(" A B C D E F G H J K L M N C
/* row 19 to 17 */
for (i = 0; i < 3; i++)
{
ii = 19 - i;
printf("%2d",ii);

Viewing a Breakpoint

Once your breakpoints have been set, you can view them in the Breakpoint window. In this step you examine breakpoints in the Breakpoint window.



Viewing Breakpoints (continued)

To View Breakpoints:

- Choose **Show Breakpoints** from the **Window** menu.

  The Breakpoint window appears with a list of breakpoints identified by file and line number.

NOTE: You can double-click any breakpoint listed in the Breakpoint window to jump immediately to the breakpoint in the Browser window.

Setting a Temporary Breakpoint

You set a temporary breakpoint to stop program execution once, then have MW Debug automatically remove the breakpoint when program execution continues. In this step you set a temporary breakpoint in the `showbord.c` file.

To Set a Temporary Breakpoint:

1. Identify where you want to set the breakpoint.

   Locate line #45 (`printf("`) in the file `showbord.c`.

2. Alt/Option+click the breakpoint dash next to the line.

   A red breakpoint circle appears briefly, indicating the presence of a breakpoint.

3. Run the code.

   Execution resumes automatically until reaching the temporary breakpoint. Execution stops and the breakpoint is removed.

Setting a Conditional Breakpoint

You set conditional breakpoints to stop program execution only when a specific condition is met. In this step you set a conditional breakpoint within the `showbord.c` file.

To Set a Conditional Breakpoint:

1. Set a breakpoint in the program.

   We'll use the breakpoint on line #49 previously. See Setting a Breakpoint for details on setting a breakpoint.

2. Open the Breakpoint window.

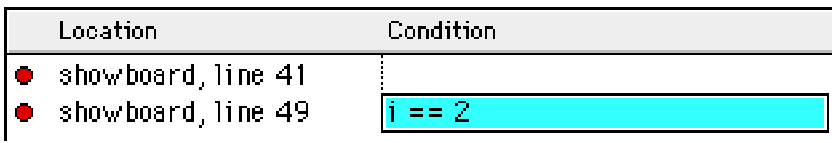
   See Opening and Closing the Breakpoint Window for details on opening the Breakpoint window.

3. Double-click the condition field next to the breakpoint.

   An outline and text insertion point appears in the condition field.

more…

Setting a Conditional Breakpoint (continued)



4. Enter the condition to halt program execution.

   You can enter an expression into the condition field by typing or dragging a condition to the field. Try entering "`i == 2`".

5. Run the code, **OR**
   Type the keyboard shortcut Ctrl/Command+R.

   The program and willnto stop at this breakpoint until the condition is true. Run to continue with the debugging session.

To Disable a Breakpoint

You can disable a breakpoint in the Breakpoints window. Whenyou do, execution does not stop, but the breakpoint remains. This is useful if you want to use the breakpoint later on. You won't have to reset it at the right spot. You can also use a disabled breakpoint as a marker to jump to a location in the code.

To Disable a Breakpoint:

- Click the red breakpoint circle in the Breakpoints window.

  Click "showboard, line 41" to disable the breakpoint. If you run the program now, it will run uninterrupted by this breakpoint.

Saving Breakpoints

You can save breakpoints between debugging sessions. Alternately, you can disable breakpoint saving to ensure that you start each debugging session with a clean breakpoint slate. In this step you enable the Save Breakpoints option.

To Save Breakpoints:

1. Choose **Preferences** from the **Edit** menu.
2. Click the **Settings & Displa**y tab.
3. Enable the **Save Breakpoints** option checkbox.
4. Click **OK**.

Clearing a Breakpoint

You can remove a breakpoint when you are finished with it so that program execution is no longer halted at that spot.

To Clear a Breakpoint

1. Locate the breakpoint to clear.
2. Click the red breakpoint circle next to the line.

   The breakpoint circle is replaced by the breakpoint dash marker.

To Clear All Breakpoints:

- Choose **Clear All Breakpoints** from the **Control** menu.

  The breakpoints are removed from the project files. If you run the project now, it will run uninterrupted.

Summary

This completes the Using Breakpoints for Debugging stage of this tour.

In this stage you learned how to:

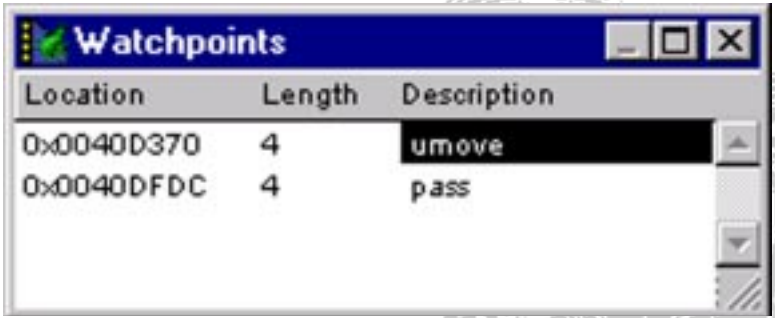
- Set normal, temporary, and condition breakpoints
- View breakpoints
- Disable and clear breakpoints
- Save breakpoints between debugging sessions

Setting normal, temporary, or conditional breakpoints is just one of the skills you should master in order to effectively debug projects. In ths next stage we look at another important debugging skill, that of manipulating project variables.

Using Watchpoints for Debugging

The Watchpoint window displays a list of currently set watchpoints. A watchpoint is a location or region of memory that the debugger watches, alerting you when any change is made in the area.

- Setting a Watchpoint
- Clearing a Watchpoint



Setting a Watchpoint

You set a watchpoint to add it to the watchpoint list so you'll be notified of any changes. In this step you add the `umove` variable to the watchpoint list.

To Set a Watchpoint:

1. Click a variable name in the **Globals** or **Variables** pane.

   Click "`umove`" at the bottom of the Globals list. The variable name is highlighted.

2. Choose **Set Watchpoint** from the **Data** menu.

   The variable `umove` is marked with a red underline and appears in the Watchpoint window.

TIP: You can also add a variable to the watchpoint list by click and dragging it to the Watchpoint window.

Clearing a Watchpoint

You clear a watchpoint to remove it from the watchpoint list. In this step you clear a watchpoint variable.

To Clear a Watchpoint:

1. Click the watchpoint variable.

   The watchpoint variable is highlighted.

2. Choose **Clear Watchpoint** from the **Data** menu.

   The variable disappears from the Watchpoint window and the red underline removed from the variable name.

Summary

This completes the Using Watchpoints for Debugging stage of this tour.

In this stage you learned how to:

- Set and clear watchpoint variables

Using the Watchpoint window to monitor a host of different memory locations and alert you when something changes can be a real boon when debugging software. Let's continue with Handling Variables.

Using Variables for Debugging

Observing and modifying the value of program variables is important during a debugging session. You've already learned about displaying variables in Touring the Variables Window, now let's see how MW Debug enables you to modify program variables.

Topics covered in this stage include:

- Changing Program Variables

Changing Program Variables

During a debugging session it's possible to change variables in order to continue with the session. In this step you modify a variable in the `gnu_go.mcp` project.

To Change a Variable:

1. Double-click the variable value.

   Double-click the "`umove`" value in the Browser 's Variable pane. The variable field is outlined and its contents highlighted.

2. Enter the new value.

   The variable is changed to the new value.

NOTE: Changing variables can be dangerous. The Debugger allows you to set a variable to any valid setting for the variable's data type. For example, it does not prevent you from setting a pointer to nil and crashing the program.

Summary

This completes the Using Variables for Debugging stage of this tour.

In this stage you learned how to:

- Modify a variable

Conclusion

Congratulations!

You have completed the basic MW Debug tutorial. You have met the main windows of MW Debug, and learned how to use them when debugging projects.

If you have not already done so, you may wish to complete the other tutorials on the CodeWarrior IDE, the Project Manager, and the Editor.

Technical Support

If you should encounter any difficulties with this guided tour, contact Metrowerks Technical Support for assistance. Please use the appropriate report form located in the Release Notes folder on the CD. They can be reached at:

`support@metrowerks.com`

Copyright Information

Metrowerks CodeWarrior copyright ©1993–1998 by Metrowerks Inc. and its licensors. All rights reserved.

Documentation stored on the compact disk(s) may be printed by licensee for personal use. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from Metrowerks Inc.

Metrowerks, the Metrowerks logo, CodeWarrior, and Software at Work are registered trademarks of Metrowerks Inc. PowerPlant and PowerPlant Constructor are trademarks of Metrowerks Inc.

All other trademarks and registered trademarks are the property of their respective owners.

ALL SOFTWARE AND DOCUMENTATION ON THE COMPACT DISK(S) ARE SUBJECT TO THE LICENSE AGREEMENT ON THE CD.

Revision Code: 1/14/98-lft